On the Computational Complexity of Head Movement and Affix Hopping

On the Computational Complexity of Head Movement and Affix Hopping

Miloš Stanojević

School of Informatics
University of Edinburgh
11 Crichton Street, Edinburgh, UK
m.stanojevic@ed.ac.uk

Abstract. Head movement is a syntactic operation used in most generative syntactic analyses. However, its computational properties have not been extensively studied. Stabler (2001) formalises head movement in the framework of Minimalist Grammars by extending the item representation to allow for easy extraction of the head. This work shows that Stabler's representation is in fact suboptimal because it causes higher polynomial parsing complexity. A new algorithm is derived for parsing head movement and affix hopping by changing the kinds of representations that the parser deals with. This algorithm has much better asymptotic worst-case runtime of $\mathcal{O}(n^{2k+5})$. This result makes parsing head movement and affix hopping computationally as efficient as parsing a single phrase movement.

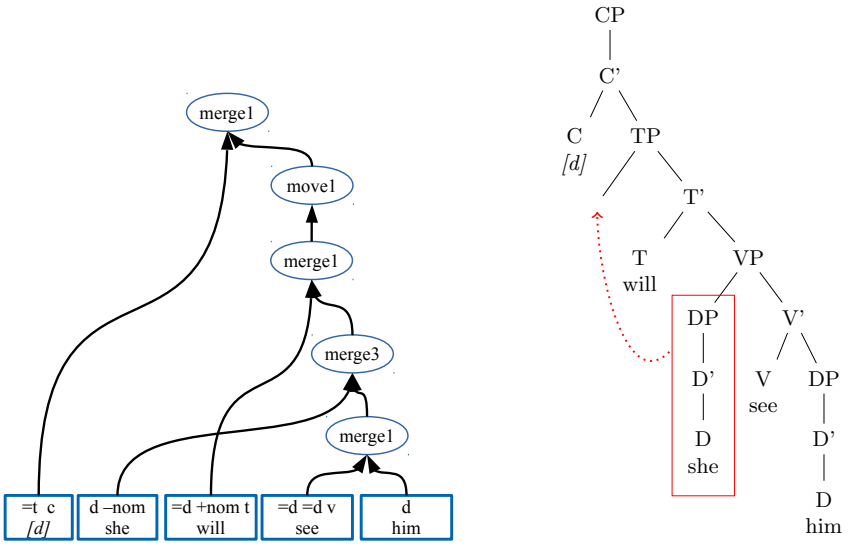
Keywords: Minimalist Grammars · Parsing · Head Movement · Affix Hopping.

1 Introduction

Minimalist Grammars (MG) [26] are a formalisation of Chomsky's Minimalist Program [4]. MGs rely on only two basic operations MERGE and MOVE. MERGE is a binary function that combines two constituents into a single constituent, while MOVE is a unary operation that takes one sub-constituent and reattaches it to the specifier position at the root of the partially constructed tree. An example MG derivation with MERGE and MOVE is shown in Figure 1a for the declarative sentence "*[d]* she will meet him" where *[d]* is a null declarative complementiser. Figure 1b shows the X-bar structure that is a byproduct of the MG derivation and can be computed deterministically. Here MERGE combines constituents that are not necessarily adjacent to each other, while MOVE raises the subject DP from spec-VP to spec-TP so that it can check the nominative case feature.

MERGE is a single function but it is often easier to view it as different subfunctions over non-overlapping domains. For instance, merge1 is applied in the case of complement attachment, while merge2 attaches a specifier. The same holds for MOVE: move1 moves a phrase to its final landing site, while move2 moves a constituent that is going to move again later. The type of movement

done by move1 and move2 is often called phrase movement because it is applied to a maximal projection (XP). However, phrasal movement is not the only type of movement used in Minimalist syntax. In addition to phrasal movement, all minimalist textbooks [1, 2, 22, 25] also discuss head movement. Head movement can be triggered when a selecting lexical head merges with its complement. This operation extracts the head of the complement and adjoins it to the selecting head. It can adjoin the complement's head to the left or right of the selecting head depending on the type of feature that triggers the head movement.

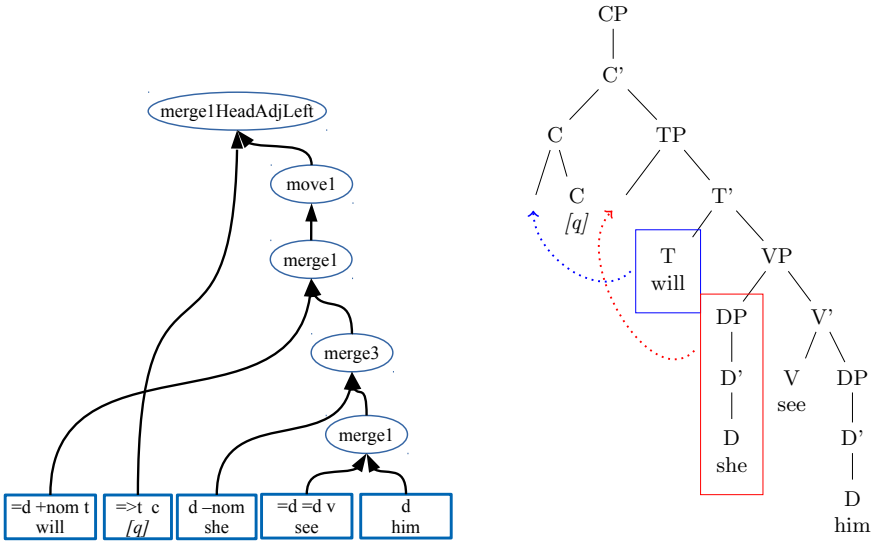


(a) derivation tree (b) X-bar tree

Fig. 1: Example without head movement

The most typical example of head movement in English is Subject-Auxiliary inversion in yes-no questions. Figure 2 shows the previous example sentence turned into a question by using a different null complementiser *[q]* for forming yes-no questions. What is different between *[d]* and *[q]* is the type of the feature that each uses to select the tense phrase: *[d]* uses a simple selector =t while *[q]* uses =>t which, in addition to selecting the tense phrase, also extracts the tense head "will" and adjoins it to the left of the complementiser head.

The original version of MG published in [26] had both phrase and head movement. There have since been many variations on MG proposed, some of which are reviewed in [29] including a simpler version that does not include head movement. This simple version of MG is very convenient for formal analysis and parsing. We will call this version *succinct MG* or MG^S, and the original version of MG with head movement MG^H.



(a) derivation tree (b) X-bar tree

Fig. 2: Example with head movement

Many phenomena that appear to require head movement can in fact be expressed with other means like remnant or rightward movement [18, 28]. Also, MG with and MG without head movement are weakly equivalent [21, 27]. Still, if we are interested in the structures that explain the derived word order using head movement we will need a parser specifically tailored to parsing with head movement. The practical need for that kind of parser is even more evident with the recent construction of MGbank [34] and the first wide-coverage MG parser [36], which both use head movement in almost all of their derivations.

The published work on Minimalist Parsing can be divided into two categories. The first is transition based parsing which is usually of higher relevance to psycho-linguists as a more likely model of human sentence processing [8, 9, 12, 13, 17, 30–33]. However, these models use no dynamic programming and therefore have exponential runtime complexity. This makes them impractical for actual parsing and inappropriate for studying the theoretical computational complexity properties of MGs.

The second type of MG parsers are those that use dynamic programming and are usually expressed as deductive systems. These parsers run in polynomial time and are guaranteed to find a parse if it exists. The first work of that sort is Harkema's CKY parser [10, 11] for MG without head movement. Stabler [27] showed a simple extension of Harkema's parser that enabled it to account for head movement. Harkema's parser for MG^S had computational time complexity of $\mathcal{O}(n^{4k+4})$ where k is a number of distinct phrase movement types. Stabler's extension for head movement raises complexity to $\mathcal{O}(n^{4k+12})$ because of the

additional spans that are required to keep the head of a constituent available for extraction by head movement. If we interpret $\mathcal{O}(n^4)$ as the price of a single phrase mover, this would mean that the price of having head movement is equivalent to having two phrase movement types.

A more modern parser for MG^S is presented in [7] which lowers the time complexity from Harkema's $\mathcal{O}(n^{4k+4})$ to $\mathcal{O}(n^{2k+3})$. This result is in fact the same algorithm as Harkema's (done through conversion to IRTG) but with more accurate computational analysis. If we apply this revised analysis to Stabler's head movement algorithm we get worst-case parsing complexity of $\mathcal{O}(n^{2k+9})$. Here, the price of each phrase movement is quadratic meaning that head movement now costs the same as having three distinct phrase mover types.

An interesting special type of MG^H is MG_0^H which is an MG that has only head movement without any phrase movement. This MG is more expressive than CFG but less expressive than TAG [21]. Parsing this MG using Stabler's algorithm with new analysis would take $\mathcal{O}(n^9)$ in the worst case. Intuitively, there should be a better algorithm than this because the worst case complexity of TAG, which is more expressive, is just $\mathcal{O}(n^6)$.

This paper presents a new and more efficient algorithm for parsing the full formulation of MG that contains head movement. This greater efficiency is accomplished with a more compact representation of the parse items and inference rules adapted for that compact representation. The parser's worst-case computational complexity is $\mathcal{O}(n^{2k+5})$. In computational terms, this makes parsing head movement as easy as parsing a single phrase movement. In the special case of MG_0^H we get a $\mathcal{O}(n^5)$ parser which is lower than TAG parsing complexity, exactly as we would expect.

2 MG without Head Movement

A Succinct Minimalist Grammar MG^S [29] is formally defined with a tuple $G = \langle \Sigma, B, Lex, c, \{\textsc{merge},\textsc{move}\}\rangle$, where Σ is the **vocabulary**, B is a set of **basic features**, Lex is a finite **lexicon** (as defined just below), $c \in B$ is the **start category**, and MERGE and MOVE are the generating functions. The basic features of the set B are concatenated with prefix operators to specify their roles, as follows:

categories, **selectees** $= B$
selectors $= \{=f \mid f \in B\}$
licensees $= \{-f \mid f \in B\}$
licensors $= \{+f \mid f \in B\}$

Let F be the set of role-marked **features**, that is, the union of the categories, selectors, licensors and licensees. Let $T = \{::, :\}$ be two **types**, indicating "lexical" and "derived" structures, respectively. Let $C = \Sigma^* \times T \times F^*$ be the set of **chains**. Let $E = C^+$ be the set of **expressions**; intuitively, an expression is a chain together with its "moving" sub-chains, if any. All expressions have to respect Shortest Movement Constraint (SMC) which states that no two chains in an expression can have the same initial feature. The functions MERGE and MOVE are defined in

Fig. 3. The **lexicon** $Lex \subset \Sigma^* \times \{::\} \times F^*$ is a finite set. The set of all **structures** that can be derived from the lexicon is $S(G) = \text{closure}(Lex, \{\textsc{merge}, \textsc{move}\})$. The set of **sentences** $L(G) = \{s \mid s \cdot c \in S(G) \text{ for some type } \cdot \in \{:, ::\}\}$, where c is the "start" category.

The functions *merge1*, *merge2* *merge3* are special cases of MERGE corresponding respectively to complement merge, specifier merge and merge of an element that is going to move in the future and therefore needs to be kept as a separate chain. Functions *move1* and *move2* are special cases of MOVE corresponding respectively to the movement of an element that is landing and to the movement of an element that will move again in the future. All functions in Fig. 3 have pairwise disjoint domains.

$$\frac{s \ ::= f\gamma \qquad\qquad t \ \cdot \ f, \alpha_1, \ldots, \alpha_k}{st \ : \ \gamma, \alpha_1, \ldots, \alpha_k}\text{merge1}$$

$$\frac{s \ := f\gamma, \iota_1, \ldots, \iota_l \qquad\qquad t \ \cdot \ f, \alpha_1, \ldots, \alpha_k}{ts \ : \ \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k}\text{merge2}$$

$$\frac{s \ \cdot \ = f\gamma, \iota_1, \ldots, \iota_l \qquad\qquad t \ \cdot \ f\delta, \alpha_1, \ldots, \alpha_k}{s \ : \ \gamma, \iota_1, \ldots, \iota_l, t : \delta, \alpha_1, \ldots, \alpha_k}\text{merge3}$$

$$\frac{s \ : \ +f\gamma, \iota_1, \ldots, \iota_l, t : -f, \alpha_1, \ldots, \alpha_k}{ts \ : \ \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k}\text{move1}$$

$$\frac{s \ : \ +f\gamma, \iota_1, \ldots, \iota_l, t : -f\gamma, \alpha_1, \ldots, \alpha_k}{s \ : \ \gamma, \iota_1, \ldots, \iota_l, t : \gamma, \alpha_1, \ldots, \alpha_k}\text{move2}$$

Fig. 3: Succinct MG. All rules are subject to SMC.

2.1 Parsing with Succinct MG

Most MG parsers are based on the *parsing as deduction* paradigm [24]. In *parsing as deduction* the parser maintains two data structures: a chart and an agenda. These data structures contain *items* that represent a set of derivation trees that share their topmost expression. The chart contains items that are already proved by the parser. The agenda contains items that could in the future combine with items from the chart to prove new items. Parsing starts with the agenda containing all the *axioms* (items that are true without the need for a proof) and an empty chart. When an item is popped out of the agenda, the parser tries to combine it with all the elements in the chart in an attempt to prove new items. For each new item, the parser first checks if it is present in the chart. If it is in the chart, the parser just discards it[1] because that item is already proved. If it is

[1] Or adds one more backpointer if we want all possible derivation trees instead of a single tree.

not present in the chart, the parser adds it both to the chart and to the agenda. Parsing stops when either the newly created item is the goal item, in which case parsing is successful, or when the agenda becomes empty, in which case parsing has failed because the sentence is not part of the language.

This description is enough to use this method for recognition. To turn it into a parser, it is sufficient to modify the chart data structure in such a way that each item in it contains a list of backpointers to the items that were used to derive it. When the goal item is constructed, it is enough to follow backpointers to find the full derivation tree.

The first parser for the succinct version of MG was presented by Harkema [10, 11]. The items of this parser are equivalent to the MG^S expressions except that instead of strings they contain spans of the strings in the sentence that is being parsed. The axioms of the parser are lexical entries for each word in the sentence with the string replaced by its span in the sentence. Inference rules are exactly the same as in the definition of MERGE and MOVE from Fig. 3.

Harkema's analysis of that algorithm is as follows. The maximal number of items in the chart (its space complexity) is n^{2k+2} because each item contains maximally $k+1$ spans (due to SMC) and each span has 2 indices in range $[0, n]$. In the worst case we will need to pop out n^{2k+2} items from the agenda. The parser needs to check for each of those popped items whether there is a proved item in the chart that could combine with it. In the worst case the number of items in the chart is n^{2k+2}. Therefore the worst-case complexity is $\mathcal{O}(n^{2k+2} \cdot n^{2k+2}) = \mathcal{O}(n^{4k+4})$.

As shown by Fowlie and Koller [7], this analysis was too pessimistic. Through conversion of MG^S to Interpreted Regular Tree Grammars (IRTG) they demonstrated that MG^S can be parsed in $\mathcal{O}(n^{2k+3})$. The same result can be obtained by converting MG^S to MCFG using Michaelis' algorithm [20] and then parsing with some well optimised MCFG parser.

However, conversion to any of these formalisms is not necessary to get efficient MG parsing. It is enough just to implement an efficient lookup in the chart. Optimising for feature lookup is not enough because it will improve only the constants that depend on the grammar. To get asymptotic improvement the lookup needs to be optimised on the item indices instead. For instance, if we have an item that as its main span has $(2, 3)$ and as its initial feature a selector, we know that *merge1* inference rule can combine it only with items whose main span start with 3. If we organise the items in the chart in a way that we can efficiently lookup all items that have particular properties, for instance "all items whose main span starts with 3", then the parsing complexity will be lower. If the popped item has m movers then we know that the item that combines with it certainly does not have more than $k-m$ movers due to SMC constraint. In the case of *merge1* and *merge2* we also know one of the indices of the main span, a fact that reduces the number of possible items that could merge even further. We can calculate complexity by summing over the computations for each possible value of m as follows $\mathcal{O}(\sum_{m=0}^{k} n^{2m+2} \cdot n^{2(k-m)+1}) = \mathcal{O}(n^{2k+3})$. This is a result for *merge1* and *merge2*, which turn out to be more expensive than *merge3*,

that does not need to apply concatenation of strings and for that reason can be done in $\mathcal{O}(n^{2k+2})$. Move rules do not need to consult the chart since they are unary rules, but we could still calculate the total number of times they would be applied as $\mathcal{O}(n^{2k+1})$ for *merge1* and $\mathcal{O}(n^{2k+2})$ for *merge2*.

Parsing as deduction systems are essentially logic programs which are evaluated bottom-up (forward chaining) with tabling [6, 15]. For those kinds of programs there is a much simpler way of calculating complexity that is based only on counting unique indices in the antecedents of the most complex inference rule [5, 19]. Clearly in total there cannot be more than k phrasal movers on the antecedent side because the consequent needs to respect SMC. Furthermore, for concatenation rules, i.e. *merge1* and *merge2*, we know that the main spans share at least one index, which leaves us with 3 unique indices for main spans and makes the complexity of those rules $\mathcal{O}(n^{2k+3})$. These rules are also the most complex inference rules and therefore the worst-case complexity of the whole algorithm is $\mathcal{O}(n^{2k+3})$. We will use this method of complexity analysis for parsing algorithms in the rest of the paper.

3 MG with Head Movement

The first description of how to parse MG with head movement was presented by Stabler [27]. It is based on modifying the MG expressions/items in such a way as to split the main string into three sub-parts: the head string s_h, the specifier string s_s and the complement string s_c. The reason for this splitting is to make the head string available for extraction by head movement. MG rules also needed modification to work with this representation. Modifications for *merge2*, *move1* and *move2* are trivial because they cannot trigger head movement. Rules for *merge1* and *merge3* got two additional versions that can trigger head movement and adjoin the complement's head to the selecting head to the left, in case of feature =>f, or to the right, in case of feature <=f.

Stabler's inference rules for MG with head movement[2] are shown in Fig. 4 together with a new complexity analysis for each of the rules calculated by the method of counting indices for each rule. The most complex rule is *merge2* which makes the whole algorithm run in $\mathcal{O}(n^{2k+9})$.

4 Improved Parsing of MG with Head Movement

Stabler's formulation is very compact, but it misses some generalisations that would make the number of indices smaller. For instance, if we take rule merge2 we can see that spans t_s and t_h always share one index because they are concatenated. The same holds for t_h and t_c. This means that the selector needs to

[2] Stabler's inference rules had a small mistake for merge3left and merge3right for allowing the possibility of a head constituent being non-lexical. The correct version of inference rules is presented in this paper but it can also be found in [21]. The correction is crucial for the more efficient parsing algorithm.

$$\frac{\epsilon, s, \epsilon \ ::= f\gamma \qquad t_s, t_h, t_c \ \cdot \ f, \alpha_1, \ldots, \alpha_k}{\epsilon, s, t_s t_h t_c \ : \ \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1} \quad \mathcal{O}(n^{2k+6})$$

$$\frac{\epsilon, s, \epsilon \ ::<= f\gamma \qquad t_s, t_h, t_c \ \cdot \ f, \alpha_1, \ldots, \alpha_k}{\epsilon, st_h, t_s t_c \ : \ \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1right} \quad \mathcal{O}(n^{2k+6})$$

$$\frac{\epsilon, s, \epsilon \ ::=> f\gamma \qquad t_s, t_h, t_c \ \cdot \ f, \alpha_1, \ldots, \alpha_k}{\epsilon, t_h s, t_s t_c \ : \ \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1left} \quad \mathcal{O}(n^{2k+6})$$

$$\frac{s_s, s_h, s_c \ := f\gamma, \iota_1, \ldots, \iota_l \qquad t_s, t_h, t_c \ \cdot \ f, \alpha_1, \ldots, \alpha_k}{t_s t_h t_c s_s, s_h, s_c \ : \ \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k} \quad \text{merge2} \quad \mathcal{O}(n^{2k+9})$$

$$\frac{s_s, s_h, s_c \ \cdot \ = f\gamma, \iota_1, \ldots, \iota_l \qquad t_s, t_h, t_c \ \cdot \ f\delta, \alpha_1, \ldots, \alpha_k}{s_s, s_h, s_c \ : \ \gamma, \iota_1, \ldots, \iota_l, t_s t_h t_c : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3} \quad \mathcal{O}(n^{2k+8})$$

$$\frac{\epsilon, s, \epsilon \ ::<= f\gamma \qquad t_s, t_h, t_c \ \cdot \ f\delta, \alpha_1, \ldots, \alpha_k}{\epsilon, st_h, \epsilon \ : \ \gamma, t_s t_c : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3right} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{\epsilon, s, \epsilon \ ::=> f\gamma \qquad t_s, t_h, t_c \ \cdot \ f\delta, \alpha_1, \ldots, \alpha_k}{\epsilon, t_h s, \epsilon \ : \ \gamma, t_s t_c : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3left} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s_s, s_h, s_c \ : \ +f\gamma, t : -f, \alpha_1, \ldots, \alpha_k}{ts_s, s_h, s_c \ : \ \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{move1} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{s_s, s_h, s_c \ : \ +f\gamma, \iota_1, \ldots, \iota_l, t : -f\gamma, \alpha_1, \ldots, \alpha_k}{s_s, s_h, s_c \ : \ \gamma, \iota_1, \ldots, \iota_l, t : \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{move2} \quad \mathcal{O}(n^{2k+6})$$

Fig. 4: Stabler's inference rules for MG with head movement together with their computational complexity.

visit n^2 constituents in the chart that would produce absolutely the same result because the two indices that are shared between the components t_s, t_h and t_c are disappearing in the consequent. One could try to reduce this problem by having a unary inference rule that packs all the main components of t before they are combined with merge2 akin to the fold transformation of logic programs [6, 14, 23]. If we push this operation to its limits we would get $\mathcal{O}(n^{2k+7})$.

To obtain an even lower complexity we need to change the representation of items. Do we need all the three components t_s, t_h and t_c from Stabler's formulation? If we look at the rules in Fig. 4 we can see that when an item is selected by a merge operation, its components are concatenated either as $t_s t_h t_c$ if the head is not extracted or as t_h and $t_s t_c$ if the head is extracted. This highlights a simple, tautological, fact: every head will either be extracted with head movement or it won't. In case it does not participate in head movement, like in the succinct version of MG, there is no need to keep 3 spans to represent projections of that head. It can all be done with a single span and by that reduce the number of indices.

If, on the other hand, the head does participate in head movement then we need only two spans: one for the head s_h and one for the concatenation of specifier and complement s_{sc} because we know with certainty that they will be concatenated after the head is extracted.

If we knew whether the head will move or not the parsing algorithm could be improved significantly, but how can we know if the head will move? The good aspect of chart based parsers is that we do not need to know that ahead of time.

$$\frac{s \overset{\text{A}}{::} = f\gamma \qquad t \overset{\text{A}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{st \overset{\text{A}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1A} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s, \epsilon \overset{\text{B}}{::} = f\gamma \qquad t \overset{\text{A}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{s, t \overset{\text{B}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1B} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s \overset{\text{A}}{::} <= f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{st_h t_{sc} \overset{\text{A}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1rightA} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s, \epsilon \overset{\text{B}}{::} <= f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{st_h, t_{sc} \overset{\text{B}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1rightB} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{s \overset{\text{A}}{::} => f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{t_h st_{sc} \overset{\text{A}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1leftA} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s, \epsilon \overset{\text{B}}{::} => f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{t_h s, t_{sc} \overset{\text{B}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1leftB} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{s \overset{\text{A}}{:} = f\gamma, \iota_1, \ldots, \iota_l \qquad t \overset{\text{A}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{ts \overset{\text{A}}{:} \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k} \quad \text{merge2A} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s_h, s_{sc} \overset{\text{B}}{:} = f\gamma, \iota_1, \ldots, \iota_l \qquad t \overset{\text{A}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{s_h, ts_{sc} \overset{\text{B}}{:} \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k} \quad \text{merge2B} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{s \overset{\text{A}}{\cdot} = f\gamma, \iota_1, \ldots, \iota_l \qquad t \overset{\text{A}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{s \overset{\text{A}}{:} \gamma, \iota_1, \ldots, \iota_l, t : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3A} \quad \mathcal{O}(n^{2k+2})$$

$$\frac{s_h, s_{sc} \overset{\text{B}}{\cdot} = f\gamma, \iota_1, \ldots, \iota_l \qquad t \overset{\text{A}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{s_h, s_{sc} \overset{\text{B}}{:} \gamma, \iota_1, \ldots, \iota_l, t : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3B} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s \overset{\text{A}}{::} <= f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{st_h \overset{\text{A}}{:} \gamma, t_{sc} : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3rightA} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s, \epsilon \overset{\text{B}}{::} <= f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{st_h, \epsilon \overset{\text{B}}{:} \gamma, t_{sc} : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3rightB} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s \overset{\text{A}}{::} => f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{t_h s \overset{\text{A}}{:} \gamma, t_{sc} : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3leftA} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s, \epsilon \overset{\text{B}}{::} => f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{t_h s, \epsilon \overset{\text{B}}{:} \gamma, t_{sc} : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3leftB} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s \overset{\text{A}}{:} +f\gamma, \iota_1, \ldots, \iota_l, t : -f, \alpha_1, \ldots, \alpha_k}{ts \overset{\text{A}}{:} \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k} \quad \text{move1A} \quad \mathcal{O}(n^{2k+1})$$

$$\frac{s_h, s_{sc} \overset{\text{B}}{:} +f\gamma, \iota_1, \ldots, \iota_l, t : -f, \alpha_1, \ldots, \alpha_k}{s_h, ts_{sc} \overset{\text{B}}{:} \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k} \quad \text{move1B} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s \overset{\text{A}}{:} +f\gamma, \iota_1, \ldots, \iota_l, t : -f\gamma, \alpha_1, \ldots, \alpha_k}{s \overset{\text{A}}{:} \gamma, \iota_1, \ldots, \iota_l, t : \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{move2A} \quad \mathcal{O}(n^{2k+2})$$

$$\frac{s_h, s_{sc} \overset{\text{B}}{:} +f\gamma, \iota_1, \ldots, \iota_l, t : -f\gamma, \alpha_1, \ldots, \alpha_k}{s_h, s_{sc} \overset{\text{B}}{:} \gamma, \iota_1, \ldots, \iota_l, t : \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{move2B} \quad \mathcal{O}(n^{2k+4})$$

Fig. 5: New inference rules for MG with head movement

We can just encode both variations of the items as axioms and let the parser combine them accordingly. Let us refer to the items whose head *must not* be extracted as items of type A, and items whose head *must* be extracted as items of type B. Type A items will have a single span, just like in succinct MG, while type B items will have two spans: s_h for the head and s_{sc} for specifier-complement. This reduces the space complexity from Stabler's $\mathcal{O}(n^{2k+6})$ to $\mathcal{O}(n^{2k+4})$.

The axioms of the new parser will for each word w_i contain an entry of type A: $w_i \overset{\text{A}}{::} \gamma$ and of type B: $w_i, \epsilon \overset{\text{B}}{::} \gamma$. These two cover both possible cases of w_i eventually being extracted and not being extracted by the head movement. All MG rules need to be modified accordingly, but the modification is very simple. We have exactly two rules for each rule of Stabler's parser. This is because every rule can have items of type A or type B as its main antecedent item (the item that has selector or licensor as initial feature).

The type of the non-main (selected) item depends on the MG operation: in case we use the head movement we know that the selected item is of type B, otherwise it is of type A. The type of the consequent item is determined by the main antecedent item: the head of the main antecedent item and the head of the consequent is the same and therefore the same constraint on the head movement (whether the head must or must not be extracted) has to stay unchanged.

This gives us the rules of inference shown in Fig. 5. The maximal complexity comes from *merge1rightB*, *merge1leftB* and *merge2B* which make the whole parsing in the worst-case $\mathcal{O}(n^{2k+5})$. This makes the computational price of parsing head movement $\mathcal{O}(n^2)$ which is the same as phrase movement. The number of rules is double the rules of Stabler, but they all have disjoint domains and can still be treated as a two operations MERGE and MOVE. Derivation trees that result from this parsing approach are isomorphic to the derivation trees of Stabler's parser with the only difference in labels of operations containing additional letter A or B.

In case the grammar does not have features for the head movement, we can exclude axioms of type B. This automatically makes parsing MG without head movement $\mathcal{O}(n^{2k+3})$ without doing any transformation to IRTG or MCFG.

5 ATB Head Movement

One interesting variation of head movement is Across-the-Board (ATB) head movement. This variation is not part of Stabler's original formalisation, but is of both theoretical and practical importance. If we accept that in interrogative sentences tense undergoes head movement to adjoin to the complementiser head, then in the case of the coordination of two tense phrases the same tense head has to be simultaneously extracted from both. An example sentence is "Who [does]$_T$ John _ like and Mary _ hate" (which also features ATB phrase movement of *who*).

A formalisation of ATB head movement, as it is used in MGbank, is given in [35]. The inference rules for ATB head movement are similar to ATB Phrase Movement rules from [16]. The formalism of MGbank has special features for

coordination that are located on coordinating conjunction and are marked with ˆ=f. There are two inference rules of relevance here. The first one combines coordinating conjunction with the right conjunct. In the new representation of items the coordinating conjunction will be of type A since its head is not going to undergo head movement, but both the left and the right conjunct will be of type B. The first inference rule has the following form in the more compact representation:

$$\frac{s \overset{\text{A}}{::} \hat{}=f \quad \hat{}=f \quad \gamma \qquad\qquad t_h, t_{sc} \overset{\text{B}}{:} f, \alpha_1, \ldots, \alpha_k}{t_h, st_{sc} \overset{\text{B}}{:} \hat{}=f \ \gamma, \alpha_1, \ldots, \alpha_k}$$

The second inference rule combines the result of the first rule with the left conjunct. Because this is ATB head movement we know that heads and all the moving chains have to unify between two antecedents. That gives us the following inference rule.

$$\frac{t_h, s_{sc} \overset{\text{B}}{:} f, \alpha_1, \ldots, \alpha_k \qquad\qquad t_h, t_{sc} \overset{\text{B}}{:} \hat{}=f \ \gamma, \alpha_1, \ldots, \alpha_k}{t_h, s_{sc}t_{sc} \overset{\text{B}}{:} \gamma, \alpha_1, \ldots, \alpha_k}$$

Even though the last rule uses two items of type B, the complexity is still low $\mathcal{O}(n^{2k+5})$ because the two antecedents share the same head.

6 Affix Hopping

Affix hopping [3] is a morphosyntactic operation similar to head movement and in some sense its opposite. Affix hopping can be interpreted as a downward head movement where the head of the selector is moving to adjoin to the complement's head. The main motivation for this rule, as is apparent from its name, is to move the tense affix to the verb in languages like English where V-to-T head movement cannot occur. For instance, in "John __ really like [-s]$_T$ Mary" the tense affix "-s" is often assumed to have undergone the affix hopping to adjoin to the main verb stem "like".

$$\frac{\epsilon, s, \epsilon \ :: \approx\!> f\gamma \qquad t_s, t_h, t_c \ \cdot \ f, \alpha_1, \ldots, \alpha_k}{\epsilon, \epsilon, t_s t_h s t_c \ : \ \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1HopRight} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{\epsilon, s, \epsilon \ :: \approx\!< f\gamma \qquad t_s, t_h, t_c \ \cdot \ f, \alpha_1, \ldots, \alpha_k}{\epsilon, \epsilon, t_s s t_h t_c \ : \ \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1HopLeft} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{\epsilon, s, \epsilon \ :: \approx\!> f\gamma \qquad t_s, t_h, t_c \ \cdot \ f\delta, \alpha_1, \ldots, \alpha_k}{\epsilon, \epsilon, \epsilon \ : \ \gamma, t_s t_h s t_c : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3HopRight} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{\epsilon, s, \epsilon \ :: \approx\!< f\gamma \qquad t_s, t_h, t_c \ \cdot \ f\delta, \alpha_1, \ldots, \alpha_k}{\epsilon, \epsilon, \epsilon \ : \ \gamma, t_s s t_h t_c : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3HopLeft} \quad \mathcal{O}(n^{2k+3})$$

Fig. 6: Stabler's inference rules for MG with affix hopping.

Affix hopping was formalised in Stabler's paper on head movement [27] with the inference rules shown in Fig. 6 where a special selector feature $\approx\!\!>\ f$ or $<\!\!\approx f$ is used to trigger affix hopping. Owing to our adoption of a different item representation to get a faster head movement parser, these affix hopping rules cannot be directly supported. For head movement we have exploited the fact that the main strings of every item are concatenated either as t_h, t_{sc} or t_{shc}. However, affix hopping has additional string combinations like t_{hc} and t_{sh}.

To account for those additional options we create two more types of items. Items of type C are items that must adjoin some head via affix hopping to the right of the head and that is why they have two main strings: a specifier-head string t_{sh} and a complement string t_c. Items of type D are similar except that they account for affix hopping to the right of the head. Items of type C and D do not trigger affix hopping but only accept (host) affix that has hopped. The new inference rules for affix hopping are shown in Fig. 7.

$$\frac{s \overset{\text{A}}{::} \approx\!\!> f\gamma \qquad t_{sh}, t_c \overset{\text{C}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{t_{sh}st_c \overset{\text{A}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \text{ merge1HopRight} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s \overset{\text{A}}{::} <\!\!\approx f\gamma \qquad t_s, t_{hc} \overset{\text{D}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{t_s st_{hc} \overset{\text{A}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \text{ merge1HopLeft} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s \overset{\text{A}}{::} \approx\!\!> f\gamma \qquad t_{sh}, t_c \overset{\text{C}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{\epsilon \overset{\text{A}}{:} \gamma, t_{sh}st_c : \delta, \alpha_1, \ldots, \alpha_k} \text{ merge3HopRight} \quad \mathcal{O}(n^{2k+2})$$

$$\frac{s \overset{\text{A}}{::} <\!\!\approx f\gamma \qquad t_s, t_{hc} \overset{\text{D}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{\epsilon \overset{\text{A}}{:} \gamma, t_s st_{hc} : \delta, \alpha_1, \ldots, \alpha_k} \text{ merge3HopLeft} \quad \mathcal{O}(n^{2k+2})$$

Fig. 7: Improved inference rules for MG with affix hopping.

We allow the selector constituent to be only of type A because its head will not be able to undergo head movement later. This is a stricter definition than Stabler's because the latter allows the empty string (which is not a head of the new constituent but only a replacement for the real head) to participate in head movement. This modification does influence the set of the derivations that could be built by the parser but in a good way: it does not make sense for an affix to undergo affix hopping downwards and then head movement upwards (or fake head movement of an empty string).

Similarly, we do not allow the selecting item to be of type C or D because after the affix has hopped, its slot will be empty, so it does not make sense for another affix to hop to its original place. Affixes move to attach to some overt word and there is none at this slot.

The rules in Fig. 7 show how items of type C and D are used for affix hopping. But it still remains to show how items of type C and D are built. To build items

$$\frac{s, \epsilon \overset{\text{C}}{::} = f\gamma \qquad t \overset{\text{A}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{s, t \overset{\text{C}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1C} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s, \epsilon \overset{\text{C}}{::} <= f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{st_h, t_{sc} \overset{\text{C}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1rightC} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{s, \epsilon \overset{\text{C}}{::} => f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{t_h s, t_{sc} \overset{\text{C}}{:} \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{merge1leftC} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{s_{sh}, s_c \overset{\text{C}}{:} = f\gamma, \iota_1, \ldots, \iota_l \qquad t \overset{\text{A}}{\cdot} f, \alpha_1, \ldots, \alpha_k}{ts_{sh}, s_c \overset{\text{C}}{:} \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k} \quad \text{merge2C} \quad \mathcal{O}(n^{2k+5})$$

$$\frac{s_{sh}, s_c \overset{\text{C}}{:} = f\gamma, \iota_1, \ldots, \iota_l \qquad t \overset{\text{A}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{s_{sh}, s_c \overset{\text{C}}{:} \gamma, \iota_1, \ldots, \iota_l, t : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3C} \quad \mathcal{O}(n^{2k+4})$$

$$\frac{s, \epsilon \overset{\text{C}}{::} <= f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{st_h, \epsilon \overset{\text{C}}{:} \gamma, t_{sc} : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3rightC} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s, \epsilon \overset{\text{C}}{::} => f\gamma \qquad t_h, t_{sc} \overset{\text{B}}{\cdot} f\delta, \alpha_1, \ldots, \alpha_k}{t_h s, \epsilon \overset{\text{C}}{:} \gamma, t_{sc} : \delta, \alpha_1, \ldots, \alpha_k} \quad \text{merge3leftC} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s_{sh}, s_c \overset{\text{C}}{:} +f\gamma, \iota_1, \ldots, \iota_l, t : -f, \alpha_1, \ldots, \alpha_k}{ts_{sh}, s_c \overset{\text{C}}{:} \gamma, \iota_1, \ldots, \iota_l, \alpha_1, \ldots, \alpha_k} \quad \text{move1C} \quad \mathcal{O}(n^{2k+3})$$

$$\frac{s_{sh}, s_c \overset{\text{C}}{:} +f\gamma, \iota_1, \ldots, \iota_l, t : -f\gamma, \alpha_1, \ldots, \alpha_k}{s_{sh}, s_c \overset{\text{C}}{:} \gamma, \iota_1, \ldots, \iota_l, t : \gamma, \alpha_1, \ldots, \alpha_k} \quad \text{move2C} \quad \mathcal{O}(n^{2k+4})$$

Fig. 8: Additional inference rules for building items of type C that must host affix hopping in the later part of the derivation.

of type C and D, we need to add them to the agenda as axioms for all lexical items (just as was done for items of type B) and to use additional inference rules that are variations of rules for items of type A and B from Fig. 5. The additional rules for items of type C are shown in Fig. 8. Similar rules are trivial to make for items of type D.

The rules merge1leftC and merge3leftC may at first appear somewhat surprising. They move the head of the complement to the left of the specifier-head string making a complex head-specifier-head. This may appear to break the rules of head movement which state that the head adjoins to another head and there cannot be any phrase in between them. However, this is not a problem in this case. Since we know that head movement can be triggered only by a lexical item we can be certain that there is no specifier in its specifier-head string, so the final result of the concatenation is a head-head complex.

7 Conclusion

The main motivation for this parser is lowering the worst-case complexity of parsing MG that contains head movement, ATB head movement and affix hopping. Given the recent appearance of the new dataset with MG derivation trees [34] which contains head movement in every derivation, this algorithm is likely to be not only of theoretical but also of practical significance.

MGbank has lead to the first wide coverage Minimalist parser [36]. This is a neural network based parser that uses A* search with Harkema's inference rules and Stabler's approach to head movement. A* improves the best and average-case scenario but the worst-case stays the same $\mathcal{O}(n^{4k+12})$ which is $\mathcal{O}(n^{28})$ for the MGbank grammar. With the algorithm proposed in this paper the overall worst-case complexity will be reduced to $\mathcal{O}(n^{13})$. More importantly, it will potentially also improve average-case complexity because of the more optimal lookup.

The complexity of the parser presented here is $\mathcal{O}(n^{2k+5})$ which is just $\mathcal{O}(n)$ bigger than its space complexity. Further asymptotic improvements would probably require either finding some more compact item representation or abandoning the *parsing as deduction* approach and using some alternative approach akin to Valiant style parsing.

Acknowledgement

I am grateful to John Torr and the three anonymous reviewers for comments that have greatly improved this paper. This work was supported by ERC H2020 Advanced Fellowship GA 742137 SEMANTAX grant.

References

1. Adger, D.: Core syntax: A minimalist approach, vol. 33. Oxford University Press Oxford (2003)
2. Carnie, A.: Syntax: A generative introduction, 3rd edition. John Wiley & Sons (2013)
3. Chomsky, N.: Syntactic structures. Mouton, The Hague (1957)
4. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge, Massachusetts (1995)
5. Dowling, W.F., Gallier, J.H.: Linear-time algorithms for testing the satisfiability of propositional horn formulae. The Journal of Logic Programming **1**(3), 267 – 284 (1984)
6. Eisner, J., Blatz, J.: Program transformations for optimization of parsing algorithms and other weighted logic programs. In: Wintner, S. (ed.) Proceedings of FG 2006: The 11th Conference on Formal Grammar. pp. 45–85. CSLI Publications (2007)
7. Fowlie, M., Koller, A.: Parsing minimalist languages with interpreted regular tree grammars. In: Proceedings of the 13th International Workshop on Tree Adjoining Grammars and Related Formalisms. pp. 11–20 (2017)

8. Gerth, S.: Memory Limitations in Sentence Comprehension: A Structural-based Complexity Metric of Processing Difficulty, vol. 6. Universitätsverlag Potsdam (2015)
9. Graf, T., Monette, J., Zhang, C.: Relative clauses as a benchmark for minimalist parsing. Journal of Language Modelling **5**(1), 57–106 (2017)
10. Harkema, H.: A Recognizer for Minimalist Grammars. In: In Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000). pp. 111–122. Springer Verlag (2000)
11. Harkema, H.: A Recognizer for Minimalist Languages. In: Bunt, H., Carroll, J., Satta, G. (eds.) New Developments in Parsing Technology (Volume 23 of the series Text, Speech and Language Technology). pp. 251–268. Springer Netherlands, Dordrecht (2005)
12. Hunter, T.: Left-corner parsing of minimalist grammars. Tech. rep., UCLA (2017), forthcoming
13. Hunter, T., Stanojević, M., Stabler, E.: The active-filler strategy in a move-eager left-corner Minimalist Grammar parser. In: Proceedings of the 9th Workshop on Cognitive Modeling and Computational Linguistics (CMCL 2019). Association for Computational Linguistics (2019)
14. Johnson, M.: Transforming projective bilexical dependency grammars into efficiently-parsable cfgs with unfold-fold. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics. pp. 168–175. Association for Computational Linguistics (2007)
15. Kanazawa, M.: Parsing and generation as datalog queries. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics. pp. 176–183. Association for Computational Linguistics, Prague, Czech Republic (Jun 2007)
16. Kobele, G.M.: Across-the-board extraction in minimalist grammars. In: Proceedings of the Ninth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+9). pp. 113–120. Association for Computational Linguistics (2008)
17. Kobele, G.M., Gerth, S., Hale, J.: Memory resource allocation in top-down minimalist parsing. In: Morrill, G., Nederhof, M.J. (eds.) Formal Grammar. pp. 32–51. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
18. Koopman, H., Szabolcsi, A.: Verbal complexes. MIT Press, Cambridge, MA (2000)
19. McAllester, D.: On the complexity analysis of static analyses. J. ACM **49**(4), 512–537 (Jul 2002)
20. Michaelis, J.: Derivational minimalism is mildly context–sensitive. In: International Conference on Logical Aspects of Computational Linguistics. pp. 179–198. Springer (1998)
21. Michaelis, J.: Notes on the complexity of complex heads in a minimalist grammar. In: Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+ 6). pp. 57–65 (2002)
22. Radford, A.: Minimalist Syntax: Exploring the Structure of English. Cambridge University Press (2004)
23. Shepherdson, J.C.: Unfold/fold transformations of logic programs. Mathematical Structures in Computer Science **2**(2), 143157 (1992)
24. Shieber, S.M., Schabes, Y., Pereira, F.C.: Principles and implementation of deductive parsing. The Journal of Logic Programming **24** (1995)
25. Sportiche, D., Koopman, H., Stabler, E.: An introduction to syntactic analysis and theory. John Wiley & Sons (2013)
26. Stabler, E.: Derivational minimalism. In: Retoré, C. (ed.) Logical Aspects of Computational Linguistics, pp. 68–95. LNCS 1328, Springer-Verlag, NY (1997)

27. Stabler, E.: Recognizing head movement. In: de Groote, P., Morrill, G., Retoré, C. (eds.) Logical Aspects of Computational Linguistics (Volume 2099 of the series Lecture Notes in Computer Science), LACL 2001 Le Croisic, France, June 27–29, 2001 Proceedings. pp. 245–260. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
28. Stabler, E.: Comparing 3 perspectives on head movement. In: From Head Movement and Syntactic Theory, UCLA/Potsdam Working Papers in Linguistics. pp. 178–198 (2003)
29. Stabler, E.: Computational perspectives on minimalism. In: Boeckx, C. (ed.) Oxford Handbook of Linguistic Minimalism, pp. 617–641. Oxford University Press, Oxford (2011)
30. Stabler, E.: Top-Down Recognizers for MCFGs and MGs. In: Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics. pp. 39–48. Association for Computational Linguistics, Portland, Oregon, USA (June 2011)
31. Stabler, E.: Two models of minimalist, incremental syntactic analysis. Topics in Cognitive Science **5**(3), 611–633 (2013)
32. Stanojević, M., Stabler, E.: A Sound and Complete Left-Corner Parsing for Minimalist Grammars. In: Proceedings of the Eight Workshop on Cognitive Aspects of Computational Language Learning and Processing. pp. 65–74. Association for Computational Linguistics (2018)
33. Stanojević, M.: Minimalist grammar transition-based parsing. In: International Conference on Logical Aspects of Computational Linguistics, LACL. pp. 273–290. LNCS 10054, Springer (2016)
34. Torr, J.: Constraining mgbank: Agreement, l-selection and supertagging in minimalist grammars. In: Proceedings of the 56th Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics, Melbourne, Australia (2018)
35. Torr, J., Stabler, E.: Coordination in minimalist grammars. In: Proceedings of the 12th Annual Workshop on Tree-Adjoining Grammars and Related Formalisms, TAG+ (2016)
36. Torr, J., Stanojević, M., Steedman, M., Cohen, S.: Wide-Coverage Neural A* Parsing for Minimalist Grammars. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Florence, Italy (2019)